Language, Library and Technology Choices

Python
The language I am most familiar with out of the two choices.

Flask
A minimal and less-opinionated web framework suited for smaller projects. Also has support for SQLAlchemy natively, as well as caching.

SQLAlchemy
A SQL library and ORM which for this simple CRUD application is useful for querying as it can take care of creating and executing the queries for you, which includes securing the query from vulnerable attacks and many other convenient features. Also it helps mirror the schema in code if the model is defined in the code and can help maintain consistency between application code and database schema. If the schema were to grow more complex, it would be beneficial to begin writing queries manually alongside in a hybrid approach for flexibility as well as performance and forego letting the ORM define the tables.

Schema
A useful library to define schemas and subsequent validations for anything including JSON and simple variables. With this, validating request JSON data and request path arguments was very easy.

Pytest
The testing framework that is feature-rich like parametirization and that I am more comfortable with.

Postgres
As this is a simple CRUD application, a typical SQL DBMS is most appropriate as it suits the rigid model. Additionally, the most supported open source DBMS. It creates the indexes we need right out of the box as well.

Docker
Docker allows the software to run on any platform, which means running it is as simple as writing one command if defined properly. In this case, I have tested that successfully.

Code and Database Design Choices

With the codebase, I tried to create a modular and flexible structure. Everything to do with each entity is under its own folder, which is pulled up to the __init__ and app files above for instantiating. The exception to this are the flask extensions which are defined in the top level extensions folder but that are actually pulled down so that they are defined separate from the application allowing flexibility to decouple in the future.

With the database design, I chose Postgres because it makes for very simple definitions which are adapted by SQLAlchemy. Every column is TEXT or DATE, keeping it simple. The only complex part is the foreign key from Books to Authors which I chose to be ON DELETE CASCADE so that there are no inconsistencies if an author is deleted and their book remains wich would violate the Foreign Key relationship. Also, I chose the relationship as 1:M for now, though it should be M:N to reflect real scenarios.

Performance Tuning

I chose to implement caching, primarily because for such a simple application it seemed the most sensible option. Also because Postgres already creates the indexes necessary for this application. For now the cache timeout is 10 seconds which is quite short, and it should be increased as needed.

With millions of records, some ways to improve performance beyond further caching and intelligent indexing would be query optimization like only selecting certain fields as needed, upgrading hardware like the resources and network and finally moving to a distributed solution.